2. Въведение в програмирането

Проф. д-р Емил Хаджиколев

1. Компютърно програмиране;
2. Нива на абстракция в езиците за програмиране (поколения ЕП);
3. Парадигми за програмиране;
4. Среди за разработка;
5. Видове програми и приложения.

Компютърно програмиране

Компютърно програмиране = програмиране

- Програмирането е **процес на създаване на компютърна програма**;
- Програмата **решава някаква задача или по-сложен проблем**;
- Решението се описва във вид на **команди, които** могат да **се изпълняват автоматизирано** (от компютър) и които са написани на език за програмиране;
- Програмирането е част от процеса по разработка на софтуер.

Основни дейности в софтуерния процес (1)

Софтуерният процес включва множество **дейности** като:

- **Анализ на проблема и изисквания към решението му –** възможно е да има множество решения, но едно да е по-подходящо.
- **Описание на проблема** – с цел съгласуване изискванията на поръчителя и това което изпълнителя (човек или фирма) е разбрал.
- **Разработване на проект** за реализацията – при по-сложни проблеми създаването на проект е задължително… В проекта се описват използваните:
  - ***архитектури*** – отделните модули и компоненти на приложението, и начините на интеграцията им;
  - ***технологии*** – стандартни готови решения – библиотеки, програми или по-сложни приложения;
  - ***техники за програмиране*** (не винаги) – специални начини за писане на кода на програмата – стил, шаблони (за дизайн), специфични изисквания…
  - и др.

Основни дейности в софтуерния процес (2)

Още дейности:

- Предварително проектиране на по-сложни алгоритми (ако има такива).
- **Разработване на алгоритми** – написването (кодирането) им на език за програмиране – това включва разработката както на алгоритми решаващи отделни задачи, така и на цялостното решение.
- **Тестване** на алгоритмите, модулите, компонентите и цялостната реализация…
- **… и** съответно **отстраняване на грешките**.
- Създаване на **документация за потребителите** на софтуера.
- Поддръжка – промени по кода, администриране (при необходимост)…
- И др.

Езици за програмиране

- Езиците за програмиране (ЕП) са **изкуствени езици**.
- (Както естествените езици ЕЕ) ЕП **притежават правила за описание на различни видове езикови конструкции** (наричани команди, операции или инструкции – съответни на изречения в ЕЕ), разбираеми от компютъра (процесора).
- **Алгоритмите се описват като последователност от команди на ЕП**.
- Примерни видове алгоритми: за изчисление, работа с данни, управление и работа с периферни устройства – монитори, принтери, клавиатури, мишки, сензорни екрани и мн. др.

Компютърни езици

- Езици за програмиране – с тях се правят програми (описват се алгоритми).
- Декларативни езици:
  - Уеб езици, като: HTML, CSS, XML…
  - За работа с бази от данни: SQL…
  - и др.

Алгоритми

- **Алгоритъм: описание на последователност от действия, които решават определена задача или клас от задачи**.
- Алгоритмите може да се представят:
  - графично – по специални правила за изобразяване на различни елементи;
  - да бъдат описани на естествен език;
  - записани на компютърен език.
- За да бъдат разбираеми за компютърна система, алгоритмите трябва да бъдат написани/преведени на **машинен език** за определен вид процесори.

Нива на абстракции (поколения) на езиците за програмиране (1)

- ***Машинни езици*** (първо ниво – 1940 г. – първите модерни компютри) – програмите се описват като последователност от нули и единици (двоичен код), съдържащи директни команди (инструкции) за конкретен тип процесори. Всеки процесор съдържа стандартен основен набор от възможни инструкции, чрез които може да се създават всякакви по-сложни конструкции.
- ***Асемблерни езици*** (второ ниво – 1950 г.) – имената на инструкциите се представят чрез символни (мнемонични) кодове; данните могат да се представят като текст, десетични или шестнадесетични числа. По-лесни са за програмистите (спрямо машинните езици), но все още доста трудни.

Нива на абстракции (поколения) на езиците за програмиране (2)

- ***Високо ниво (трето) – машинно независими езици***
  - Кодът за създаване на програмите съдържа инструкции, които се доближават до естествен език.
  - ЕП има формализиран синтаксис – правила за писане на кода.
  - Относително лесно се създават програми.
  - За да бъдат разбрани от компютър, програмите трябва да бъдат преведени – транслирани – на машинен език. За целта се създават специализирани транслиращи програми – компилатори и интерпретатори – които превеждат кода до специфичен за конкретен процесор машинен език.
  - Притежават възможности за създаване на абстракции (процедури, обекти и др.), които могат да бъдат използвани многократно.
  - В това ниво са по-широко използваните езици за програмиране, като някои от тях притежават и елементи от следващото ниво.

Нива на абстракции (поколения) на езиците за програмиране (3)

- **Четвърто ниво** – ориентирани към решението на сложни задачи в определена предметна област. Като основа за програмирането се поставя семантиката (смисъла) на проблемите които се описват. Основават се върху автоматизирането на дейности, които могат да бъдат описани чрез диаграми по специализирани нотации – диаграми за управление на работни потоци (data flow diagrams), entity relationship (в областта на базите от данни) и др. Включват среда за разработка, обектно-ориентирани езици, бази от данни и други инструменти. Съдържат множество основни елементи за съответната област и правила за работа с тях. Видове:
  - Дизайнер на бази от данни (database design tools, database modeler)
  - Генератор на доклади и справки (reports) от един или повече източници на данни;
  - Управление на данни – използват се команди за (задаване на бизнес-работна логика върху данните), статистически анализи и справки;
  - Управление на бизнес процеси.
  - И мн. др.
- **Пето ниво** – езици свързани с обработка на знания и изкуствения интелект.

Йерархичност в нивата на ЕП и програмирането (1)

- Обикновено всяко ниво предоставя базови възможности за следващото ниво.
- Тези възможности са достъпни чрез функции, които могат да бъдат достъпвани чрез името си (и параметри към тях).

- Реализацията на функциите от по-ниско ниво е скрита за функциите от по-високо ниво.
- Функциите от по-високо ниво трябва да знаят само как да използват (да се обръщат към) функциите от по-ниско ниво.
- Всяко ниво предоставя интерфейс – множество от функции – за следващото (по-високо) ниво.

Процесорни архитектури

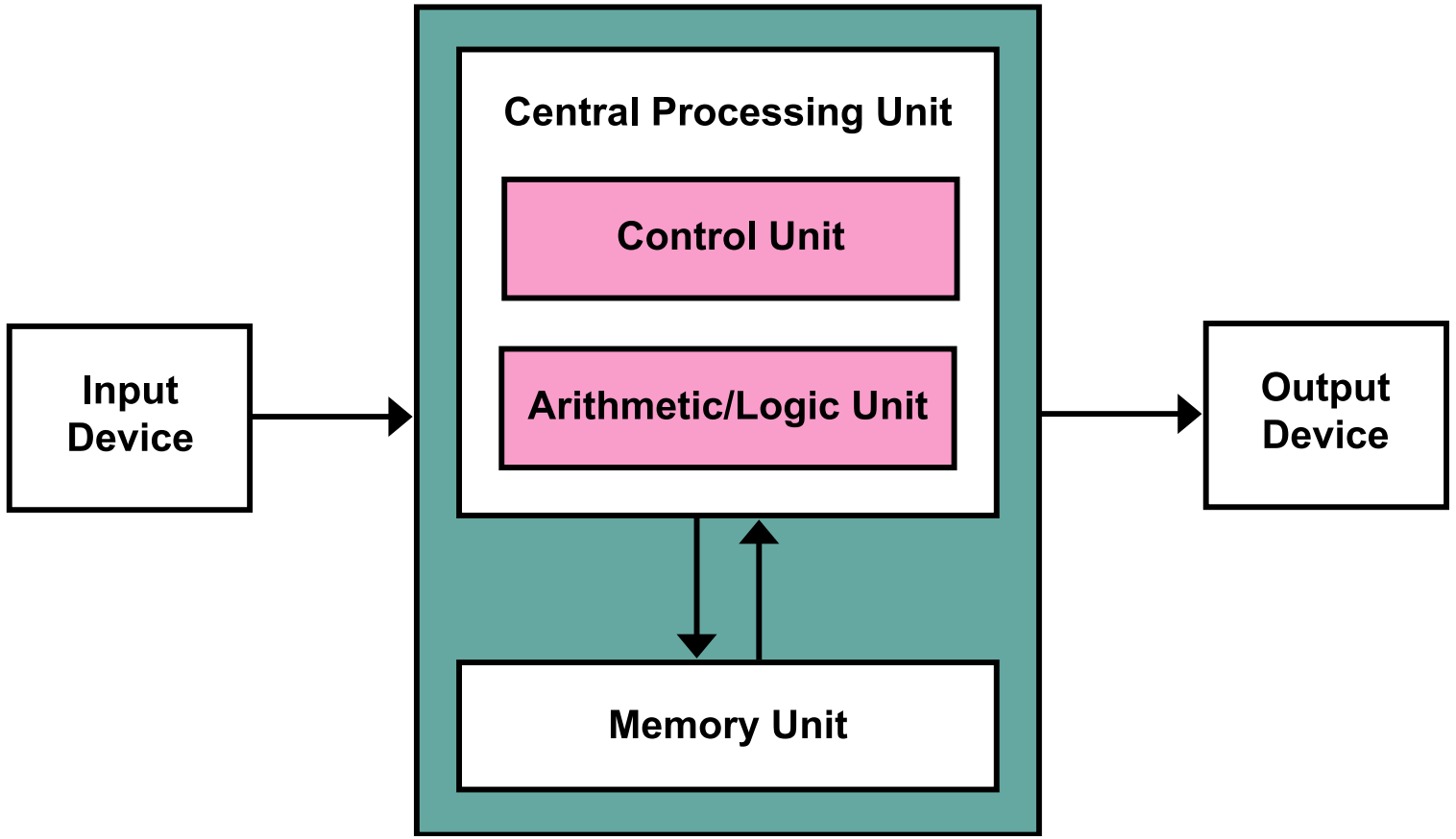
- Машинните езици са различни за процесори с различни архитектури, за различни версии на процесорите и предоставени от различни производители.
- Напр. най-често използваните процесори от обикновени потребители – Intel и AMD – имат една и съща архитектура, но процесорите от един и същи клас може да имат малки разлики в множествата от поддържани инструкции (instruction set). Тези разлики могат да се използват за решаването на специфични задачи от специфични приложения.

Основни процесорни архитектури

- x86 (за 8-, 16- и 32-битови процесори) и x86-64 (64-битови) – ползва се в процесорите на Intel (основана през 1968 г.) и AMD (1969 г.).
- ARM (Advanced RISC Machines) – за мобилни устройства и др.
- MIPS (Microprocessor without Interlocked Pipeline Stages) – за вградени системи и мрежови устройства.
- POWER: Разработена от IBM. Използва се за високопроизводителни сървъри и др.
- SPARC (Scalable Processor Architecture): Разработена от Sun Microsystems (придобита от Oracle). Използва се за сървъри и вградени системи.
- и др.

Основна концепция за съвременна компютърна архитектура

- Всички споменати архитектури на процесори са съобразени с концепцията на **Джон фон Нойман** за компютърна архитектура, публикувана 1945/46.



Създатели на съвременните цифрови компютри

- 1936 г. – **Алън Тюринг – теоретични принципи на универсална изчислителна машина – „ машина на Тюринг“**.
- 1938 г. – **Клод Шенън** за първи път демонстрира, че **двупозиционни елементи (електронни релета, лампи и др.)** могат да бъдат използвани за извършване на **математически и логически операции в двоична бройна система**.
- 1936 – 44 г. – **електро-механични машини, изградени с релета** – **Конрад Цузе** (машини Z1, Z2, Z3, Z4), **Джордж Стибиц** (BEL-1, BEL-2, BEL-5), **Хауърд Айкен** (Марк I, Марк-2).
- 1936 – 41 г. – **Джон Атанасов** и асистентът му Клифърд Бери – **първата електронна изчислителна машина (с електронни лампи, кондензатори и др.)**
- 1943 – 45 г. – **първата универсална действаща електронна изчислителна машина ENIAC** (Electronics Numerical Integrator and Computer), работеща с десетична бройна система – Мокли, Екерт и Голдстайн и мн. др.
- 1945 – **Джон фон Нойман – принципи на компютър със запаметена програма** „EDVAC“
- **Първи компютри със запаметена програма** – **EDVAC (1950 г.) , EDSAC (1949 г. – Морис Уилкс).**

Z4 – с двупозиционни електромеханични релета

https://en.wikipedia.org/wiki/Z4_(computer)

Реплика на компютъра на Атанасов-Бери

https://bg.wikipedia.org/wiki/%D0%94%D0%B6%D0%BE%D0%BD_%D0%90%D1%82%D0%B0%D0%BD%D0%B0%D1%81%D0%BE%D0%B2

Двупозиционни елементи, използвани в компютърните системи за извършване на аритметични действия с двоични числа и логически (побитови) операции върху групи от битове

1. Релета (по-съвременни) 2. Електронни лампи 3. Транзистори

Интегрални схеми, съдържащи огромен брой миниатюрни транзистори (+ диоди и резистори, а понякога кондензатори и др.)

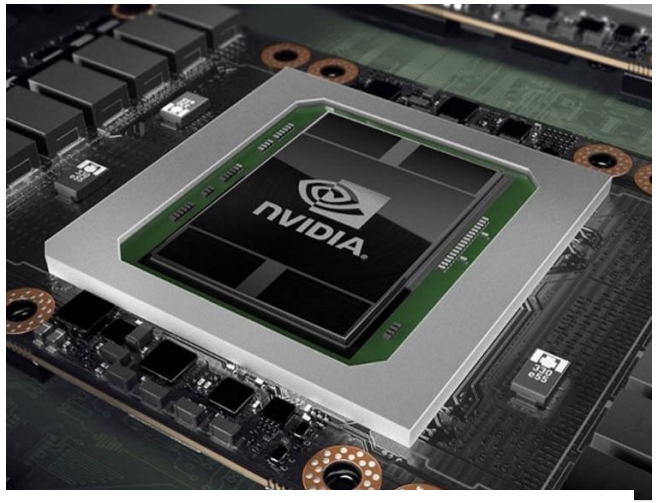
Основни принципи в архитектурата на Джон фон Нойман

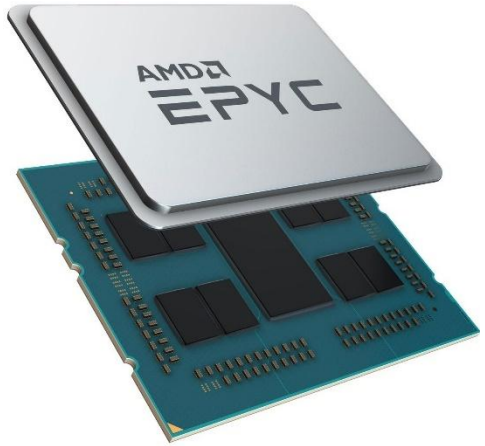
- **Централна памет (Оперативна памет):** Съхранява инструкции и данни. Достъпът до паметта става чрез адреси на клетки от паметта.
- **Процесор (CPU):** Изпълнява инструкции, взети от паметта. Състои се от
  - **АЛУ (Аритметично-логическо устройство)** за изпълнение на аритметични и логически операции върху двоични числа;
  - **Управляващ модул с регистри**, в които се записват данни.
  - **Програмен брояч**, който указва адреса на следващата инструкция, която трябва да бъде изпълнена.
- **Входно-изходни устройства:** за взаимодействие с външни обекти - клавиатура, мишка, екран...
- **Еднородност на паметта и инструкциите:** Данните и инструкциите се съхраняват и обработват по един и същи (подобен) начин.
- **Програмируемост**: Може да се изпълняват различни видове инструкции и различни програми.

Видове процесори в съвременните компютри

- CPU (Central Processing Unit) – централен процесор;
  - като част от CPU или като отделен модул може да има FPU (Floating Point Unit) – модул за ускоряване на обработката на числа с плаваща запетая;
- GPU (Graphics Processing Unit) – за ускоряване работата с графики и обработка на изображения (в игри, анимации и др.); вече и за паралелни изчисления и изкуствен интелект;
- TPU (Tensor Processing Unit) – процесор за ускоряване работата с изкуствен интелект, предлаган от Google след 2018 г.
- и др.

Процесори – съдържащи множество интегрални схеми и др. елементи

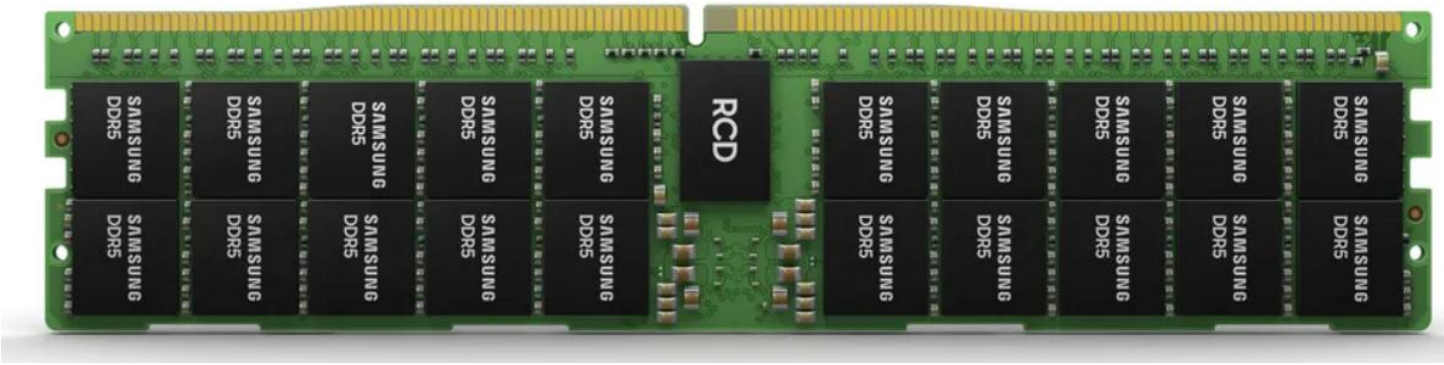




Видеокарта с GPU, вентилатори и др.

Видове памет

(На снимката: DDR5 – RAM)



- В **оперативната памет** (бърза; **RAM** – Random Access Memory) се зареждат програми (или части от тях) и данни.
- **Регистрите** са част от процесора. Те са (най-бързата) памет, в която се зареждат текущо изпълняваните инструкции, адреси и данни. Един регистър има големина 32 бита, 64 бита в съвременните (обикновени) компютри (8, 16 – при по старите).
- Във **външната памет** (бавна) – хард диск, флаш памет, CD, DVD и др. – се съхраняват дълготрайно програми и данни (в големи обеми).
- **Read-only memory (ROM)** – **памет само за четене**. Използва се за съхранение на някои програми, които не трябва да бъдат модифицирани (може, но трудно). Например програмата BIOS (Basic Input/Output System) с основни драйвери за входно-изходни устройства се зарежда при стартиране на компютъра от твърда памет и след това се зарежда операционната система.

Работа с паметта

0	1	2							n

- **Логически, паметта** (без регистрите) **се разглежда като едномерно пространство от байтове**.
- Всеки байт си има уникален пореден номер – наречен адрес.
- Обикновено, адресите се представят с шестнадесетични цифри – напр. A0BCF145 (32-битов адрес – 4 байта).
- В програмите се описват инструкции и данни (или адреси, от които се взимат данните).
  - Като се знае типа на данните (който тип е описан в програмата), намиращи се на определен адрес, може да се прочете съответната стойност – число, символ или съставен обект.
- Процесорът има достъп до всеки произволен адрес на оперативната памет.

Примери на асемблер за събиране на целите числа 2 и 3 за различни процесори (1) (ChatGPT)

- x86(32-битов):

```
mov eax, 2     ; Зареди стойността 2 в регистър eax
add eax, 3     ; Събери стойността 3 към стойността в eax
```

- x86-64 (64-битов):

```
mov rax, 2    ; Зареди стойността 2 в регистър rax
add rax, 3    ; Събери стойността 3 към стойността в rax
```

- ARM (32-битов):

```
mov r0, #2     ; Зареди стойността 2 в регистър r0
add r0, r0, #3 ; Събери стойността 3 към стойността в r0
```

- ARM64 (64-битов):

```
mov x1, #2     ; Зареди стойността 2 в регистър x1
add x1, x1, #3 ; Събери стойността 3 към стойността в x1
```

Примери на асемблер за събиране на целите числа 2 и 3 за различни процесори (2) (ChatGPT)

- MIPS (32-битов):

  ```
  li $t0, 2       ; Зареди стойността 2 в регистър $t0
  addi $t0, $t0, 3 ; Събери стойността 3 към стойността в $t0
  ```

- SPARC (64-битов):

  ```
  mov %o0, 2      ; Зареди стойността 2 в регистър %o0
  add %o0, %o0, 3 ; Събери стойността 3 към стойността в %o0
  ```

- POWER (64-битов):

  ```
  li r3, 2    ; Зареди стойността 2 в регистър r3
  li r4, 3    ; Зареди стойността 3 в регистър r4
  add r5, r3, r4 ; Събери стойностите от регистрите r3 и r4 и резултата запиши в регистър r5
  ```

Същият код на машинен език (ChatGPT)

- x86 (32-битов) – в шестнадесетичен вид

  B8 02 00 00 00 ; mov eax, 2

  83 C0 03 ; add eax, 3

- x86 (32-битов) – в двоичен вид

  10111000 00000010 00000000 00000000 00000000 ; mov eax, 2

  10000000 11000000 00000011 ; add eax, 3

(Интервалите между байтовете не съществуват в кода – сложени са за по-добра четимост от хората.)

Код на асемблер за събиране на две цели числа (намиращи се в т. нар. стека на програмата) (ChatGPT)

- X86 – шестнадесетичен формат:

```
section .text
    global _start

_start:
    ; Предполагаме, че числата 2 и 3 са вече в стека
    pop eax      ; Извади първото число от стека в регистър eax
    pop ebx      ; Извади второто число от стека в регистър ebx
    add eax, ebx  ; Събери числата и резултата запиши в eax

    ; Тук може да се използва резултата в eax по ваше усмотрение

    ; Изход от програмата
    mov eax, 1
    int 0x80
```

- x86 – двоичен формат:

```
31C0 89C3 5B 58 01D8 B801000000 CD80
```

Програма на C++ за събиране на целите числа 2 и 3 и извеждане на сумата в конзолата

```cpp
#include <iostream>
using namespace std;

int main() {
    int num1 = 2;
    int num2 = 3;
    int sum = num1 + num2;

    cout << "Sum of " << num1 << " and " << num2 << " is: " << sum << endl;

    return 0;
}
```

Парадигми на програмирането

Начините на писане на код са се изменяли във времето. Съществуват различни подходи (парадигми) за създаването на компютърен код:

- Програмиране чрез **задаване на инструкции** към процесора - машинен код и асемблерни езици
- **Процедурно програмиране;**
- **Обектно-ориентирано програмиране;**
- **Функционално програмиране;**
- **Логическо програмиране;**
- **Събитийно програмиране;**
- **Паралелно и конкурентно програмиране и др.**

Някои ЕП поддържат едновременно различни парадигми за програмиране.

Процедурно програмиране

В дисциплината „Програмиране“ разглеждаме процедурното програмиране.

- Кодът за създаване на програмите съдържа команди, които се доближават до ЕЕ (естествените езици).
- Съществуват стандартни синтактични конструкции за условия, цикли и др. (Използват се и в по-късно възникналите парадигми.)
- Решенията на проблемите (алгоритмите) се описват стъпка по стъпка.
- Част от стъпките могат се описват като подалгоритми-процедури, които могат да се извикват и изпълняват многократно.
- Процедурите работят с параметри, глобални и локални данни.

Езици: FORTRAN, ALGOL, COBOL, C, C++, Python, Perl, PHP, Visual Basic, Java, JavaScript, C# и мн. др.

Процедурно програмиране – пример

- В разгледания пример за събиране на 2 числа на езика C++
  - `int main()` е декларация на функция/процедура, в която се извършват различни действия.
  - `int` – тип на връщана стойност – цяло число. Върнатата стойност се ползва от други процедури/функции (в случая може да се ползва от други програми).
  - `main` – име на функцията.
  - в скобите след името се описват параметри. В друга версия на функцията могат да се взимат параметри от конзолата – `int main(int argc, char* argv[])`.

Обектно-ориентирано програмиране

- Програмата се описва с класове и обекти.
- Класовете описват множество от възможни същности (обект от реалния свят, понятие, процес, модел, явления...) чрез физически и функционални характеристики (реализирани като полета и методи, които описват действия върху полетата).
- Обектите са конкретни представители на класовете.
- Обектите комуникират помежду си чрез изпращане на съобщения (извиквания на методи).

Езици: C++, Python, Ruby, PHP, Visual Basic, Java, JavaScript, C# и мн. др.

Клас за работа с числа на C++, притежаващ метод за събиране

```cpp
#include <iostream>
using namespace std;

// Клас за работа с числа
class Numbers {
public:
    // Конструктор на класа
    Numbers() {}

    // Метод за събиране на две числа
    int sum(int a, int b) {
        return a + b;
    }
};
```

```cpp
int main() {
    // Създаване на обект от класа
    Numbers nums;

    int num1 = 2;
    int num2 = 3;

    // Извикване на метода за събиране
    int result = nums.sum(num1, num2);

    cout << "Sum: " << result << endl;

    return 0;
}
```

Функционално програмиране

- Алгоритмите се описват чрез функции – т. нар. ламбда (λ) функции.
- За разлика от процедурното програмиране, функциите се разглеждат като математически функции – получават параметри, задължително връщат резултат.
- Стойности на променливи и параметри могат да бъдат други функции (closure, clojure), включително анонимни функции (които нямат име).
- Често се използват рекурсивни извиквания на функции – създават се елегантни решения на сложни задачи (което рекурсивно извикване обаче изисква много системни ресурси).

Езици: Lisp, Scheme, Clojure, Haskell, Erlang, F#, Scala, Elixir, Perl, Python, PHP, C++, Java, C#, R, JavaScript и др.

Възможностите за функционално програмиране не са налични от първите версии за някои от езиците.

Функционално програмиране – пример на C++

```cpp
#include <iostream>
#include <functional>
using namespace std;

int main() {
    // Дефиниране на функционален обект add за събиране
    function<int(int, int)> add = [](int a, int b) { return a + b; };

    int num1 = 2;
    int num2 = 3;

    int sum = add(num1, num2);
    cout << "Sum: " << sum << endl;

    return 0;
}
```

Логическо програмиране

- Базира се на знания/факти и правила за тяхната обработка.
- За дадена предметна област се описват факти и правила за обектите и взаимоотношения между тях.
- Върху така създадената база от знания (семантична мрежа) се задават въпроси.

Език: Пролог и други, производни от Пролог.

Пример на Пролог за събиране на две числа

```
% Коментар: Правило за събиране на числа и извеждане на резултата.
% Има лява и дясна част разделени с „:-“. X, Y и S са параметри.
sum(X, Y) :- S is X + Y, writeln(X + Y  = S).

% Примерни заявки
:- sum(2, 3).                 % резултат: 2+3=5
:- sum(2, -3).                % резултат: 2+ -3=-1
```

При някои компилатори/интерпретатори на Пролог може да има разлики в синтаксиса – напр. заявката да е с „?-“, а не с „:-“

Събитийно програмиране

(без пример)

- Изпълнението на кода на приложението зависи от възникването на конкретни събития.
- т.е. Определени части от код се изпълняват при възникване на събития като натискане на бутон (onClick); преминаване с мишката върху елемент (onMouseOver); натискане на клавиш (onKeyPress, onKeyDown-без да е отпуснат), отпускане на клавиш (onKeyUp) и мн. др.
- Използва се в приложенията с графичен потребителски интерфейс (GUI) – мобилни приложения, десктоп приложения, уеб приложения, за вградени системи, специализирани в различни области...
- Използва се и в приложения без GUI – работа с файлове (отваряне, промяна изтриване), бази от данни, системни събития, времеви и мн. др.
- Програмистите може да създават и собствени събития и начини за обработката им.

Паралелно и конкурентно програмиране
(без пример)

- Свързани са с едновременното изпълнение на няколко задачи/процеса.
- Използват се при сървърни приложения – такива, които изпълняват услуги за множество клиентски приложения.
- Паралелни процеси могат да се изпълняват на един процесор, с редуващи се превключвания между отделните задачи.
- При множество процесорни ядра може да се постигне по-висока ефективност и производителност на някои приложения.
- При паралелното програмиране задачата се разделя на паралелни процеси (omp parallel sections в C++).
- При конкурентното програмиране се използват нишки (treads), които могат да се изпълняват
  - синхронно – последователно или;
  - асинхронно – нишките не зависят една от друга.

Декларативна и императивна парадигми за програмиране

(по-общи от разгледаните вече парадигми)

- Императивна парадигма:
  - Кодът описва алгоритми – последователности от стъпки/действия върху данни, във вид на команди, които компютърът изпълнява.
  - Езиците за програмиране (с които се създават програми) използват този подход: C++, C#, Java, Python, PHP и мн. др.
- Декларативна парадигма:
  - Кодът указва какво да се прави, а не как.
  - Не се описват алгоритми. Изпълнението на декларациите обаче се извършва от приложения, интерпретиращи кода (на декларациите).
  - Такива езици са компютърни езици, но не са езици за програмиране: HTML, XML, CSS, SQL, Prolog, езици за логическо, функционално и паралелно програмиране (само в частта за съответната парадигма) и др.

Стандарти и версии на езиците

- За някои от компютърните езици съществуват ISO стандарти – C, C++, C#, JavaScript, Prolog, SQL. Такива езици се имплементират (реализират) от различни разработчици.
- Други езици – Java, PHP и др. имат един основен разработчик.

- Версии на C++: …, C++14 (ISO/IEC 14882:2014), C++17(ISO/IEC 9899:2018), C++20 (ISO/IEC 14882:2020), C++23 (разработва се).

- В различните версии, стандарти и имплементации има разлики – съответно, проблеми в кода могат да възникнат не само заради грешки на разработчика/програмиста, а и поради използваната реализация на езика.

Интегрирани среди за разработка (1)

IDE – Integrated Development Environment

Средите за разработка улесняват множество аспекти на софтуерния процес:

- Моделиране
- Програмиране
- Тестване
- Създаване на документация
- И др.

Интегрирани среди за разработка (2)

Основни характеристики:

- работа с различни ЕП;
- управление на множество проекти;
- работа с бази от данни;
- текстов редактор на кода с вграден анализатор, който:
  - с цел по-добра четимост на кода оцветява по различен начин различните елементи от кода (величини, подпрограми, методи, ключови думи и др.);
  - по време на писане предлага автоматично довършване на думи (autocomplete) в зависимост от контекста;
  - предупреждава за възможна неправилна работа на кода, синтактични грешки и др.;
- лесно стартиране на приложенията;
- автоматизирана работа с кода – преформатиране по определени стандартни правила за писане, автоматично добавяне на елементи и др.;
- инструменти за дебъгване, работа в екип, готови шаблони за приложения и др.

Конкретни среди за разработка

- **Microsoft Visual Studio** – за работа с реализирани от Microsoft продукти – Visual C++, Visual C#, Visual F#, Visual Basic, SQL Server и др. (като JavaScript, Python)
- **NetBeans** – Java, HTML5, JavaScript, PHP, C, C++
- **Eclipse** – Java, C, C++, JavaScript, PHP, Perl, Prolog, Python, R, Ruby
- **Android Studio** – разработка на мобилни приложения
- и др.

Средите за разработка притежават възможности за добавяне на plugin-и, улесняващи създаването на специфични приложения.

Програми и приложения

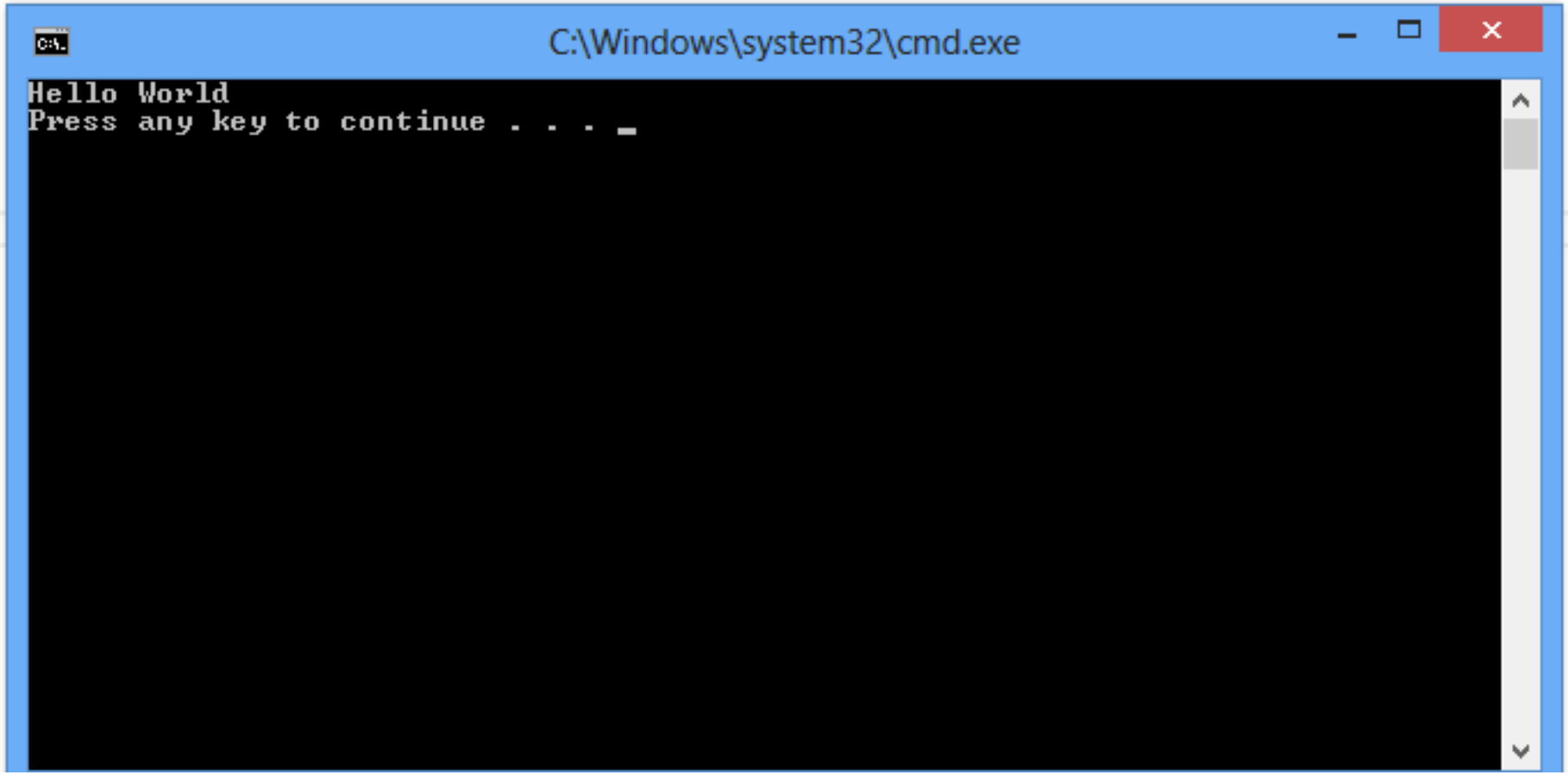
- Обикновено не се прави разлика между програма и приложение.
- **Програмите се състоят от един модул** и имат една входна точка за начало на изпълнението си. Свързани са с решаването на конкретни задачи.
- **Приложенията може да се състоят от много модули или програми**, които работят заедно. Може да имат различни входни точки. Различните модули може да са написани на различни езици.
- Приложенията (application) решават множество логически-зависими проблеми/задачи в дадена предметна област.
  - При уеб приложенията, например, работната логика може да е реализирана в самостоятелни уеб страници или услуги (всяка от които се достъпва със собствен url адрес).

Видове програми и приложения

- Конзолни;
- Десктоп приложения с графичен потребителски интерфейс (GUI);
- Уеб приложения (сайтове);
- Мобилни;
- и др.

Конзолни програми

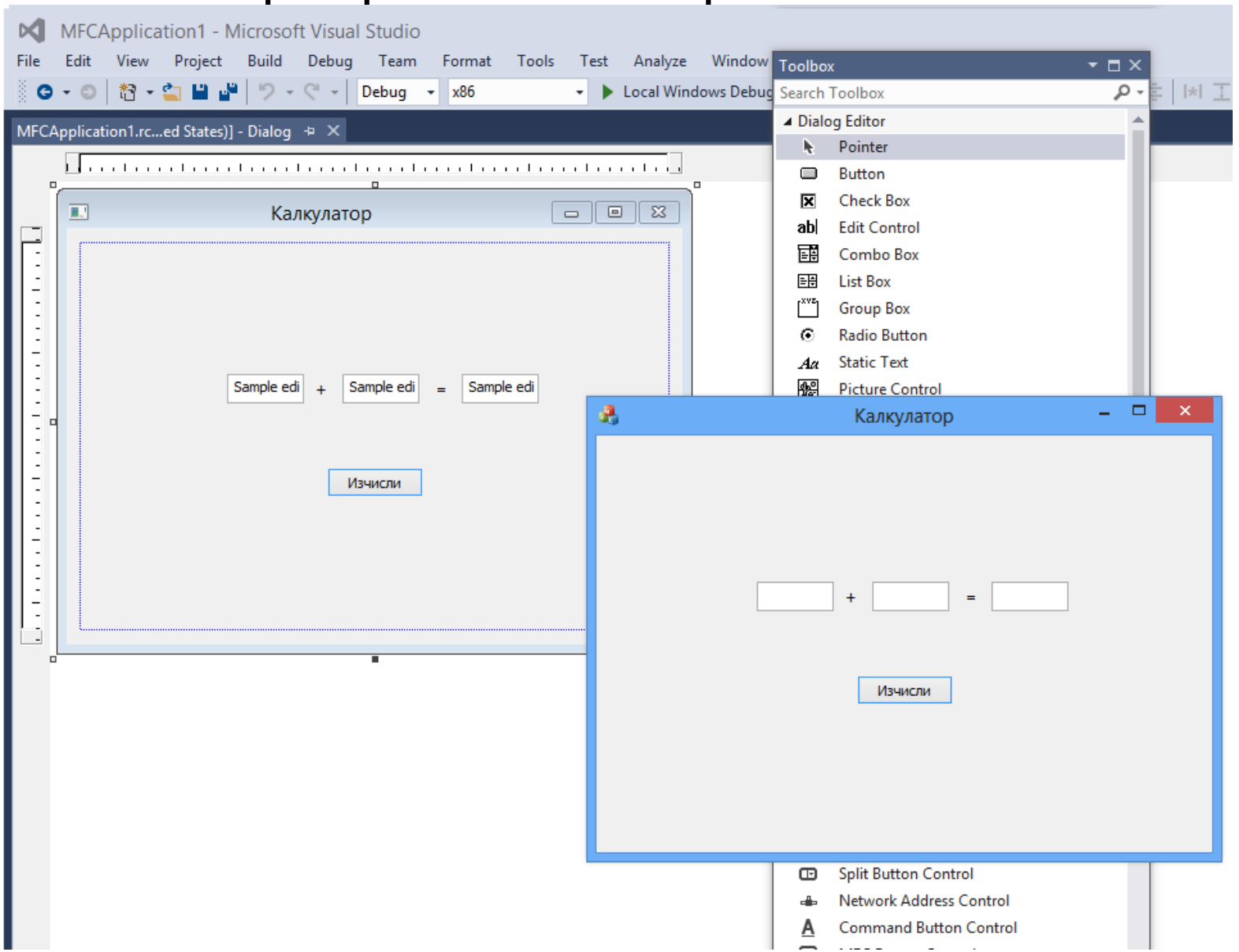
```cpp
#include <iostream>

using namespace std;

int main()
{
    cout << "Hello World\n";
    return 0;
}
```



Десктоп приложения с графичен потребителски интерфейс (GUI)

Пример, създаден с Visual Studio (MFC Application).

В средите има визуален редактор за избор на компоненти и настройване на характеристиките им, задаване на действия при събития – като, например, натискане на бутон.



Уеб и мобилни приложения

Примери!